WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, September 14, 2005

Hide?	Set Name	Query	Hit Count
	DB=PGPB; PLUR=YES; OP=ADJ		
☐	L6	717/148.ccls.	98
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L5	L4 and l1	137
☐	L4	(717/148,162-165).ccls.	755
☐	L3	717/148,162-165.ccls.	755
☐	L2	L1 and (initializ$5 with variable)	236
☐	L1	(dynamic$6 or (run time)) with compiler	2110

END OF SEARCH HISTORY

Results for "((((dynamic<in>ab) <and> (compiler<in>ab)))<and>(share<in>metada..." e-mail
Your search matched **38** of **272** documents.
A maximum of **100** results are displayed, **25** to a page, sorted by **Relevance** in **Descending** order.

Select Article Information

1. **Cacheminer: A runtime approach to exploit cache locality on SMP**
Yong Yan; Xiaodong Zhang;
Parallel and Distributed Systems, IEEE Transactions on
Volume 11, Issue 4, April 2000 Page(s):357 - 374
Digital Object Identifier 10.1109/71.850833
AbstractPlus | References | Full Text: PDF(1392 KB) IEEE JNL

2. **Physical experimentation with prefetching helper threads on Intel's hyper processors**
Kim, D.; Liao, S.S.-W.; Wang, P.H.; del Cuvillo, J.; Tian, X.; Zou, X.; Wang, H.; Girkar, M.; Shen, J.P.;
Code Generation and Optimization, 2004. CGO 2004. International Symposiun 2004 Page(s):27 - 38
Digital Object Identifier 10.1109/CGO.2004.1281661
AbstractPlus | Full Text: PDF(390 KB) IEEE CNF

3. **Extending OpenMP to support slipstream execution mode**
Ibrahim, K.Z.; Byrd, G.T.;
Parallel and Distributed Processing Symposium, 2003. Proceedings. Internatio 22-26 April 2003 Page(s):10 pp.
Digital Object Identifier 10.1109/IPDPS.2003.1213119
AbstractPlus | Full Text: PDF(278 KB) IEEE CNF

4. **Applying interposition techniques for performance analysis of OPENMP p applications**
Gonzalez, M.; Serra, A.; Martorell, X.; Oliver, J.; Ayguade, E.; Labarta, J.; Nava
Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedir International
1-5 May 2000 Page(s):235 - 240
Digital Object Identifier 10.1109/IPDPS.2000.845990
AbstractPlus | Full Text: PDF(160 KB) IEEE CNF

5. **A graph based framework to detect optimal memory layouts for improvin**
Kandemir, M.; Choudhary, A.; Ramanujam, J.; Banerjee, P.;
Parallel and Distributed Processing, 1999. 13th International and 10th Sympos and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings
12-16 April 1999 Page(s):738 - 743
Digital Object Identifier 10.1109/IPPS.1999.760558

AbstractPlus | Full Text: PDF(156 KB) IEEE CNF

6. **A memory-layout oriented run-time technique for locality optimization on**
Yong Yan; Xiaodong Zhang; Zhao Zhang;
Parallel Processing, 1998. Proceedings. 1998 International Conference on
10-14 Aug. 1998 Page(s):189 - 196
Digital Object Identifier 10.1109/ICPP.1998.708484
AbstractPlus | Full Text: PDF(96 KB) IEEE CNF

7. **Using cache optimizing compiler for managing software cache on distrib memory system**
Nanri, T.; Sato, H.; Shimasaki, M.;
High Performance Computing on the Information Superhighway, 1997. HPC A
28 April-2 May 1997 Page(s):312 - 318
Digital Object Identifier 10.1109/HPC.1997.592166
AbstractPlus | Full Text: PDF(540 KB) IEEE CNF

8. **Time-shared hybrid computers: A new concept in computer-aided design**
Howe, R.M.; Hollstien, R.B.;
Proceedings of the IEEE
Volume 60, Issue 1, Jan. 1972 Page(s):71 - 77
AbstractPlus | Full Text: PDF(2836 KB) IEEE JNL

9. **Parallel Newton type methods for power system stability analysis using l memory multiprocessors**
Chai, J.S.; Zhu, N.; Bose, A.; Tylavsky, D.J.;
Power Systems, IEEE Transactions on
Volume 6, Issue 4, Nov. 1991 Page(s):1539 - 1545
Digital Object Identifier 10.1109/59.117001
AbstractPlus | Full Text: PDF(688 KB) IEEE JNL

10. **The potential of compile-time analysis to adapt the cache coherence enf strategy to the data sharing characteristics**
Mounes-Toussi, F.; Lilja, D.J.;
Parallel and Distributed Systems, IEEE Transactions on
Volume 6, Issue 5, May 1995 Page(s):470 - 481
Digital Object Identifier 10.1109/71.382316
AbstractPlus | References | Full Text: PDF(1228 KB) IEEE JNL

11. **A notation for deterministic cooperating processes**
Mani Chandy, K.; Foster, I.;
Parallel and Distributed Systems, IEEE Transactions on
Volume 6, Issue 8, Aug. 1995 Page(s):863 - 871
Digital Object Identifier 10.1109/71.406962
AbstractPlus | References | Full Text: PDF(804 KB) IEEE JNL

12. **An efficient solution to the cache thrashing problem caused by true data**
Jin, G.; Li, Z.; Chen, F.;
Computers, IEEE Transactions on
Volume 47, Issue 5, May 1998 Page(s):527 - 543
Digital Object Identifier 10.1109/12.677228
AbstractPlus | References | Full Text: PDF(508 KB) IEEE JNL

13. **Adaptive protocols for software distributed shared memory**
Amza, C.; Cox, A.L.; Dwarkadas, S.; Li-Jie Jin; Rajamani, K.; Zwaenepoel, W.;
Proceedings of the IEEE
Volume 87, Issue 3, March 1999 Page(s):467 - 475
Digital Object Identifier 10.1109/5.747867

AbstractPlus | References | Full Text: PDF(160 KB) IEEE JNL

14. **Exploiting wavefront parallelism on large-scale shared-memory multipro**
Manjikian, N.; Abdelrahman, T.S.;
Parallel and Distributed Systems, IEEE Transactions on
Volume 12, Issue 3, March 2001 Page(s):259 - 271
Digital Object Identifier 10.1109/71.914756
AbstractPlus | References | Full Text: PDF(1640 KB) IEEE JNL

15. **The inside story on shared libraries and dynamic loading**
Beazley, D.M.; Ward, B.D.; Cooke, I.R.;
Computing in Science & Engineering [see also IEEE Computational Science a
Volume 3, Issue 5, Sept.-Oct. 2001 Page(s):90 - 97
Digital Object Identifier 10.1109/5992.947112
AbstractPlus | References | Full Text: PDF(168 KB) IEEE JNL

16. **Minimum register instruction sequencing to reduce register spills in out-superscalar architectures**
Govindarajan, R.; Hongbo Yang; Amaral, J.N.; Chihong Zhang; Gao, G.R.;
Computers, IEEE Transactions on
Volume 52, Issue 1, Jan. 2003 Page(s):4 - 20
Digital Object Identifier 10.1109/TC.2003.1159750
AbstractPlus | References | Full Text: PDF(1062 KB) IEEE JNL

17. **Design and engineering of a dynamic binary optimizer**
Duesterwald, E.;
Proceedings of the IEEE
Volume 93, Issue 2, Feb 2005 Page(s):436 - 448
Digital Object Identifier 10.1109/JPROC.2004.840302
AbstractPlus | Full Text: PDF(288 KB) IEEE JNL

18. **Design space exploration of a software speculative parallelization schem**
Cintra, M.; Llanos, D.R.;
Parallel and Distributed Systems, IEEE Transactions on
Volume 16, Issue 6, June 2005 Page(s):562 - 576
Digital Object Identifier 10.1109/TPDS.2005.69
AbstractPlus | Full Text: PDF(2408 KB) IEEE JNL

19. **Memory Management for Data Localization on OSCAR Chip Multiprocess**
Nakano, H.; Kodaka, T.; Kimura, K.; Kasahara, H.;
Innovative Architecture for Future Generation High-Performance Processors a
2004. Proceedings
12-14 Jan. 2004 Page(s):82 - 88
Digital Object Identifier 10.1109/IWIA.2004.10020
AbstractPlus | Full Text: PDF(184 KB) IEEE CNF

20. **Implementing a reconfigurable atomic memory service for dynamic netw**
Musial, P.M.; Shvartsman, A.A.;
Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th Inte
26-30 April 2004 Page(s):208
Digital Object Identifier 10.1109/IPDPS.2004.1303237
AbstractPlus | Full Text: PDF(1388 KB) IEEE CNF

21. **FlexSim simulation environment**
Nordgren, W.B.;
Simulation Conference, 2003. Proceedings of the 2003 Winter
Volume 1, 7-10 Dec. 2003 Page(s):197 - 200 Vol.1
AbstractPlus | Full Text: PDF(450 KB) IEEE CNF

☐ **22. SPARK: a high-level synthesis framework for applying parallelizing comp transformations**
Gupta, S.; Dutt, N.; Gupta, R.; Nicolau, A.;
VLSI Design, 2003. Proceedings. 16th International Conference on
4-8 Jan. 2003 Page(s):461 - 466
Digital Object Identifier 10.1109/ICVD.2003.1183177
AbstractPlus | Full Text: PDF(294 KB) IEEE CNF

☐ **23. Flexsim simulation environment**
Nordgren, W.B.;
Simulation Conference, 2002. Proceedings of the Winter
Volume 1, 8-11 Dec. 2002 Page(s):250 - 252 vol.1
Digital Object Identifier 10.1109/WSC.2002.1172892
AbstractPlus | Full Text: PDF(382 KB) IEEE CNF

☐ **24. Software environment for a multiprocessor DSP**
Kalavade, A.; Othmer, J.; Ackland, B.; Singh, K.J.;
Design Automation Conference, 1999. Proceedings. 36th
21-25 June 1999 Page(s):827 - 830
Digital Object Identifier 10.1109/DAC.1999.782150
AbstractPlus | Full Text: PDF(428 KB) IEEE CNF

☐ **25. Dynamically exploiting narrow width operands to improve processor pov performance**
Brooks, D.; Martonosi, M.;
High-Performance Computer Architecture, 1999. Proceedings. Fifth Internatior
On
9-13 Jan. 1999 Page(s):13 - 22
Digital Object Identifier 10.1109/HPCA.1999.744314
AbstractPlus | Full Text: PDF(128 KB) IEEE CNF

Results for "((dynamic<in>ab) <and> (compiler<in>ab))<and> (initializer<in>a..." e-mail

Your search matched **0** documents.

A maximum of **100** results are displayed, **25** to a page, sorted by **Relevance** in **Descending** order.

No results were found.

Please edit your search criteria and try again. Refer to the Help pages if you need assistan search.

Terms used **dynamic compiler share initializer variable** Found **87** of **160,906**

Sort results by: relevance

Display results: expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 20 of 87 Result page: **1** 2 3 4 5 next

Relevance scale

1 Static conflict analysis for multi-threaded object-oriented programs

Christoph von Praun, Thomas R. Gross

May 2003 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation**, Volume 38 Issue 5

Full text available: pdf(674.11 KB) Additional Information: full citation, abstract, references, citings, index terms

A compiler for multi-threaded object-oriented programs needs information about the sharing of objects for a variety of reasons: to implement optimizations, to issue warnings, to add instrumentation to detect access violations that occur at runtime. An Object Use Graph (OUG) statically captures accesses from different threads to objects. An OUG extends the Heap Shape Graph (HSG), which is a compile-time abstraction for runtime objects (nodes) and their reference relations (edges). An OUG specifie ...

Keywords: heap shape graph, object use graph, program analysis, race detection, representations for concurrent programs

2 Practical extraction techniques for Java

Frank Tip, Peter F. Sweeney, Chris Laffra, Aldo Eisma, David Streeter

November 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 6

Full text available: pdf(1.01 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Reducing application size is important for software that is distributed via the internet, in order to keep download times manageable, and in the domain of embedded systems, where applications are often stored in (Read-Only or Flash) memory. This paper explores extraction techniques such as the removal of unreachable methods and redundant fields, inlining of method calls, and transformation of the class hierarchy for reducing application size. We implemented a number of extraction techniques in < ...

Keywords: Application extraction, call graph construction, class hierarchy transformation, packaging, whole-program analysis

3 Multitasking without comprimise: a virtual machine evolution

Grzegorz Czajkowski, Laurent Daynés

October 2001 **ACM SIGPLAN Notices , Proceedings of the 16th ACM SIGPLAN**

conference on Object oriented programming, systems, languages, and applications, Volume 36 Issue 11

Full text available: pdf(220.97 KB) Additional Information: full citation, abstract, references, citings, index terms

The multitasking virtual machine (called from now on simply MVM) is a modification of the Java virtual machine. It enables safe, secure, and scalable multitasking. Safety is achieved by strict isolation of application from one another. Resource control augment security by preventing some denial-of-service attacks. Improved scalability results from an aggressive application of the main design principle of MVM: share as much of the runtime as possible among applications and replicate everything el ...

Keywords: Java virtual machine, application isolation, native code execution, resource control

4 Compatible genericity with run-time types for the Java programming language

Robert Cartwright, Guy L. Steele

October 1998 **ACM SIGPLAN Notices , Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 33 Issue 10

Full text available: pdf(1.97 MB) Additional Information: full citation, abstract, references, citings, index terms

The most serious impediment to writing substantial programs in the Java™ programming language is the lack of a *gentricity* mechanism for abstracting classes and methods with respect to type. During the past two years, several research groups have developed Java extensions that support various forms of genericity, but none has succeeded in accommodating general type parameterization (akin to Java arrays) while retaining compatibility with the existing. Java Virtual Machine. In thi ...

5 A practical type system and language for reference immutability

Adrian Birka, Michael D. Ernst

October 2004 **ACM SIGPLAN Notices , Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications**, Volume 39 Issue 10

Full text available: pdf(171.73 KB) Additional Information: full citation, abstract, references, index terms

This paper describes a type system that is capable of expressing and enforcing immutability constraints. The specific constraint expressed is that the abstract state of the object to which an immutable reference refers cannot be modified using that reference. The abstract state is (part of) the transitively reachable state: that is, the state of the object and all state reachable from it by following references. The type system permits explicitly excluding fields or objects from the abstract ...

Keywords: Java, Javari, const, immutability, mutable, readonly, type system, verification

6 Customization: optimizing compiler technology for SELF, a dynamically-typed object-oriented programming language

C. Chambers, D. Ungar

June 1989 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation**, Volume 24 Issue 7

Full text available: pdf(1.87 MB) Additional Information: full citation, abstract, references, citings, index terms

Dynamically-typed object-oriented languages please programmers, but their lack of static type information penalizes performance. Our new implementation techniques extract static

type information from declaration-free programs. Our system compiles several copies of a given procedure, each customized for one receiver type, so that the type of the receiver is bound at compile time. The compiler predicts types that are statically unknown but likely, and inserts ...

7 Types and persistence in database programming languages

Malcolm P. Atkinson, O. Peter Buneman

June 1987 **ACM Computing Surveys (CSUR)**, Volume 19 Issue 2

Full text available: pdf(7.91 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Traditionally, the interface between a programming language and a database has either been through a set of relatively low-level subroutine calls, or it has required some form of embedding of one language in another. Recently, the necessity of integrating database and programming language techniques has received some long-overdue recognition. In response, a number of attempts have been made to construct programming languages with completely integrated database management systems. These lang ...

8 Practicing JUDO: Java under dynamic optimizations

Michał Cierniak, Guei-Yuan Lueh, James M. Stichnoth

May 2000 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation**, Volume 35 Issue 5

Full text available: pdf(190.06 KB) Additional Information: full citation, abstract, references, citings, index terms

A high-performance implementation of a Java Virtual Machine (JVM) consists of efficient implementation of Just-In-Time (JIT) compilation, exception handling, synchronization mechanism, and garbage collection (GC). These components are tightly coupled to achieve high performance. In this paper, we present some static anddynamic techniques implemented in the JIT compilation and exception handling of the Microprocessor Research Lab Virtual Machine (MRL VM), ...

9 The architecture of Montana: an open and extensible programming environment with an incremental C++ compiler

Michael Karasick

November 1998 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering**, Volume 23 Issue 6

Full text available: pdf(1.16 MB) Additional Information: full citation, abstract, references, citings, index terms

Montana is an open, extensible integrated programming environment for C++ that supports incremental compilation and linking, a persistent code cache called a CodeStore, and a set of programming interfaces to the CodeStore for tool writers. CodeStore serves as a central source of information for compiling, browsing, and debugging. CodeStore contains information about both the static and dynamic structure of the compiled program. This information spans files, macros, declarations, function bodies, ...

Keywords: C++, conpilation, extensible systems, frameworks, incremental compilation, incremental development environments, programming environments

10 Performance monitoring: TEST: a tracer for extracting speculative threads

Michael Chen, Kunle Olukotun

March 2003 **Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization CGO '03**

Additional Information:

Full text available: pdf(1.76 MB) full citation, abstract, references, citings, index terms

Thread-level speculation (TLS) allows sequential programs to be arbitrarily decomposed into threads that can be safely executed in parallel. A key challenge for TLS processors is choosing thread decompositions that speedup the program. Current techniques for identifying decompositions have practical limitations in real systems. Traditional parallelizing compilers do not work effectively on most integer programs, and software profiling slows down program execution too much for real-time analysis. ...

11 Borrow, copy or steal?: loans and larceny in the orthodox canonical form

Anthony J. H. Simons

October 1998 **ACM SIGPLAN Notices , Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 33 Issue 10

Full text available: pdf(2.09 MB) Additional Information: full citation, abstract, references, index terms

Dynamic memory management in C++ is complex, especially across the boundaries of library abstract data types. C++ libraries designed in the orthodox canonical form (OCF) alleviate some of the problems by ensuring that classes which manage any kind of heap structures faithfully copy and delete these. However, in certain common circumstances, OCF heap structures are wastefully copied multiple times. General reference counting is not an option in OCF, since a shared body violates the intended value ...

Keywords: C++, borrowing, copy-on-write, implementation strategies, larceny, memory management, stealing, transfer of ownership

12 Type-preserving compilation of Featherweight Java

Christopher League, Zhong Shao, Valery Trifonov

March 2002 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 24 Issue 2

Full text available: pdf(378.51 KB) Additional Information: full citation, abstract, references, citings, index terms

We present an efficient encoding of core Java constructs in a simple, implementable typed intermediate language. The encoding, after type erasure, has the same operational behavior as a standard implementation using vtables and self-application for method invocation. Classes inherit super-class methods with no overhead. We support mutually recursive classes while preserving separate compilation. Our strategy extends naturally to a significant subset of Java, including interfaces and privacy. The ...

Keywords: Java, object encodings, type systems, typed intermediate languages

13 Extending Java for high-level Web service construction

Aske Simon Christensen, Anders Møller, Michael I. Schwartzbach

November 2003 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 25 Issue 6

Full text available: pdf(947.02 KB) Additional Information: full citation, abstract, references, citings, index terms

We incorporate innovations from the <bigwig> project into the Java language to provide high-level features for Web service programming. The resulting language, JWIG, contains an advanced session model and a flexible mechanism for dynamic construction of XML documents, in particular XHTML. To support program development we provide a suite of program analyses that at compile time verify for a given program that no runtime errors can occur while building documents or receiving form input, and ...

Keywords: Interactive Web services, XML, data-flow analysis

14 Lazy evaluation of C++ static constructors
Marc Sabatella
June 1992 **ACM SIGPLAN Notices**, Volume 27 Issue 6

Full text available: pdf(644.08 KB) Additional Information: full citation, abstract, index terms

Static constructors in C+ + are functions that are executed implicitly to initialize objects at run time. Although a C+ + implementation is allowed to defer the construction of objects defined in a given translation unit until the first use of any function or object defined in that translation unit, most implementations execute all static constructors for the entire program before the invocation of main(). In this paper, we describe an implementation that defers the execution of static construct ...

15 The KaffeOS Java runtime system
Godmar Back, Wilson C. Hsieh
July 2005 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 27 Issue 4

Full text available: pdf(704.30 KB) Additional Information: full citation, abstract, references, index terms

Single-language runtime systems, in the form of Java virtual machines, are widely deployed platforms for executing untrusted mobile code. These runtimes provide some of the features that operating systems provide: interapplication memory protection and basic system services. They do not, however, provide the ability to isolate applications from each other. Neither do they provide the ability to limit the resource consumption of applications. Consequently, the performance of current systems degra ...

Keywords: Robustness, garbage collection, isolation, language runtimes, resource management, termination, virtual machines

16 Abstract data types are under full control with Ada 9X
Magnus Kempe
November 1994 **Proceedings of the conference on TRI-Ada '94**

Full text available: pdf(1.23 MB) Additional Information: full citation, abstract, references, citings, index terms

Ada 83 did not provide enough control on the creation, assignment, and destruction of objects of user-defined types. This lack of control restricted type composition and prevented the full exercise of information hiding for abstract data types. Ada 9X brings new mechanisms supporting the full control of abstract data types, powerful type composition, and more extensive information hiding. With better control of abstract data types, Ada code will be easier to write, understand, maintain, and ...

17 A formal basis for architectural connection
Robert Allen, David Garlan
July 1997 **ACM Transactions on Software Engineering and Methodology (TOSEM)**, Volume 6 Issue 3

Full text available: pdf(463.23 KB) Additional Information: full citation, abstract, references, citings, index terms, review

As software systems become more complex, the overall system structure—or software architecture—becomes a central design problem. An important step toward an engineering discipline of software is a formal basis for describing and analyzing these designs. In the article we present a formal approach to one aspect of architectural design: the interactions

among components. The key idea is to define architectural connectors as explicit semantic entities. These are specified as a col ...

Keywords: WRIGHT, formal models, model-checking, module interconnection, software analysis

18 Bugs as deviant behavior: a general approach to inferring errors in systems code
Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, Benjamin Chelf
October 2001 **ACM SIGOPS Operating Systems Review , Proceedings of the eighteenth ACM symposium on Operating systems principles**, Volume 35 Issue 5

Full text available: pdf(1.53 MB) Additional Information: full citation, abstract, references, citings, index terms

A major obstacle to finding program errors in a real system is knowing what correctness rules the system must obey. These rules are often undocumented or specified in an ad hoc manner. This paper demonstrates techniques that automatically extract such checking information from the source code itself, rather than the programmer, thereby avoiding the need for a priori knowledge of system rules.The cornerstone of our approach is inferring programmer "beliefs" that we then cross-check for contradict ...

19 Environmental acquisition: a new inheritance-like abstraction mechanism
Joseph Gil, David H. Lorenz
October 1996 **ACM SIGPLAN Notices , Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, Volume 31 Issue 10

Full text available: pdf(2.40 MB) Additional Information: full citation, abstract, references, citings, index terms

The class of an object is not necessarily the only determiner of its runtime behaviour. Often it is necessary to have an object behave differently depending upon the other objects to which it is connected. However, as it currently stands, object-oriented programming provides no support for this concept, and little recognition of its role in common, practical programming situations. This paper investigates a new programming paradigm, *environmental acquisition* in the context of *object ag ...*

20 Region-based memory management in cyclone
Dan Grossman, Greg Morrisett, Trevor Jim, Michael Hicks, Yanling Wang, James Cheney
May 2002 **ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation**, Volume 37 Issue 5

Full text available: pdf(249.21 KB) Additional Information: full citation, abstract, references, citings, index terms

Cyclone is a type-safe programming language derived from C. The primary design goal of Cyclone is to let programmers control data representation and memory management without sacrificing type-safety. In this paper, we focus on the region-based memory management of Cyclone and its static typing discipline. The design incorporates several advancements, including support for region subtyping and a coherent integration with stack allocation and a garbage collector. To support separate compilation, C ...